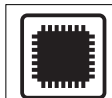
続 実設計に応用できる演算回路スキルを身につけよう

外村元伸

第5回 除算・開平・逆数・逆開平数とその演算回路設計(2) ディジット選択規則を中心に

除算・開平・逆数・逆開平数の演算器は，部分剰余を求める漸化式を利用することによって，共通に設計できます．そして，商，開平，逆数，逆開平ディジットは，これら共通のディジット選択規則を導くことによって求められます．ところが，ディジット選択規則を実際に導出することは，特に部分剰余演算にキャリ・セーブ・アダーを使う場合には結構難しく，やっかいです．そのためディジット選択規則にまつわることで，あまり知られていないことがあります．今回は，このあたりのことについて詳しく解説します．（筆者）

ディジット選択規則の導出

● 伝統的に使われている漸化式のふしぎ

1ビットずつディジットを求めていく基数2の除算において，第iステップ目の漸化式は，

$$R_{i+1} \leftarrow 2\left(R_i - q_i \cdot X\right),\ R_0 = Y \qquad \cdots(1)$$

であることを前回（2007年5月号，pp141-146の連載第4回）説明しましたが，実は多くの本や論文では，この式は使われていません．その代わりに，

$$R_{i+1} \leftarrow 2R_i - q_{i+1} \cdot X,\ R_0 = Y \qquad \cdots(2)$$

が伝統的に使われています．これは，なぜなのでしょうか．開平では，部分開平値 $Q_i = \sum_{k=0}^{i} q_k \cdot 2^{-k}$ を採用していますが，筆者は除算では，部分商 $Q_i = \sum_{k=0}^{i-1} q_k \cdot 2^{-k}$ を採用しています．総和の最大数が，iか$i-1$の違いだけですが，このちょっとしたテクニックによって，除算・開平・逆数・逆開平数の第iステップ目の漸化式が式(3)のように，同じような形式になるからです．

$$\begin{cases} \text{除算：}\ R_{i+1} \leftarrow 2\left(R_i - q_i \cdot X\right),\ R_0 = Y \\ \text{開平：}\ R_{i+1} \leftarrow 2\left(R_i - q_{i+1} \cdot \left(Q_i + q_{i+1} \cdot 2^{-(i+2)}\right)\right), \\ \qquad Q_{i+1} = Q_i + q_{i+1} \cdot 2^{-(i+1)},\ R_0 = X,\ Q_0 = 0 \\ \text{逆数：}\ R_{i+1} \leftarrow 2\left(R_i - q_i \cdot X\right),\ R_0 = 1 \\ \text{逆開平数：}\ R_{i+1} \leftarrow 2\left(R_i - q_{i+1} \cdot X\left(Q_i + q_{i+1} \cdot 2^{-(i+2)}\right)\right) \\ \qquad Q_{i+1} = Q_i + q_{i+1} \cdot 2^{-(i+1)},\ R_0 = 1 - X,\ Q_0 = 1 \end{cases} \qquad \cdots(3)$$

では，開平と同じように$Q_i = \sum_{k=0}^{i} q_k \cdot 2^{-k}$を除算でも採用してみましょう．その場合の漸化式を導くと，

$$\begin{aligned} R_{i+1} &= 2^{i+1}\left(Y - Q_{i+1}X\right) \\ &= 2^{i+1}\left(Y - Q_i X - q_{i+1} \cdot 2^{-(i+1)} X\right) \\ &= 2R_i - q_{i+1}X \end{aligned} \qquad \cdots(4)$$

となり，確かに式(2)が導かれます．この違いを意味的に考えてみます．式(1)は，現在の部分剰余R_iを用いて，商ディジットq_iを決定します．それにもとづいて部分剰余を演算し，2倍して次の部分剰余R_{i+1}とします．それに対して式(2)は，現在の部分剰余R_iを2倍してから，つまり，$2R_i$を用いて，商ディジットq_{i+1}を決定します．商ディジットq_{i+1}にもとづいて部分剰余を演算し，そのまま次の部分剰余R_{i+1}とします．ここでは，q_iのiの値は相対的に一つずれていますが，大した意味をもちません．決定的な違いは，R_iと$2R_i$で，商ディジットの選択のときに効いてきま

KeyWord
除算，開平，逆数，逆開平数，部分剰余，ディジット選択規則，漸化式，キャリ・セーブ・アダー，符号付きディジット，ディジット・アダー，SSDアダー，Simplified Signed-digit Adder

表1　ディジット選択規則

q_i	r_{-1}	r_0	r_1	$(s_2,\ c_2)$
＋1	0	1	1	＊
＋1	0	1	0	＊
＋1	0	0	1	＊
＋1	0	0	0	Not 0
0	0	0	0	0
0	1	1	1	＊
－1	1	1	0	＊
－1	1	0	1	＊
未定義	1	0	0	＊

表2　キャリ・セーブ・アダーの演算規則とその特徴をとらえる表現

＊={1, 2} ＋={0, 1}		加数			
		11	10	01	00
被加数	11	1＋	0＊	2＋	1＊
	10	0＊	0＋	1＊	1＋
	01	2＋	1＊	1＋	0＊
	00	1＊	1＋	0＊	0＋

```
  121        121
 +101       +100
 1110       1100
```

(**a**) キャリ・セーブ・アダーの初期状態からの演算結果

＊={1, 2} ＋={0, 1}		加数			
		11	10	01	00
被加数	22	2＊	2＋	1＊	1＋
	21	2＋	1＊	1＋	0＊
	20	1＊	1＋	0＊	0＋
	12	1＊	1＋	2＊	2＋
	11	1＋	0＊	2＋	1＊
	10	0＊	0＋	1＊	1＋
	02	2＊	2＋	1＊	1＋
	01	2＋	1＊	1＋	0＊
	00	1＊	1＋	0＊	0＋

(**b**) キャリ・セーブ・アダーの完全な演算結果

す．例えば式(2)の場合，$Y>X$のときは，$R_0=Y=[0.11]$b，$X=[0.10]$bの例では，$2R_0=(r_{-1},\ r_0,\ r_1)=(0,\ 1,\ 1)=[01.1]$bなので，**表1**のディジット選択規則から，$q_1=1$です．

$$R_1=[01.0]\mathrm{b}\leftarrow 2Y-q_1\cdot X=[01.1]\mathrm{b}-[0.10]\mathrm{b}=[01.0]\mathrm{b}$$
$$2R_1=\left(r_{-1},\ r_0,\ r_1\right)=(1,\ 0,\ 0)=[10.0]\mathrm{b} \quad \cdots(5)$$

から，ディジット選択規則の定義域の範囲を超えてしまい($R_1>0$なのに，$2R_1<0$となる)，部分剰余演算を続行できません．このような状態を避けるには，$Y>X$のときは，Yの値を1ビット右にシフトして調節します($Y/2<X$)．しかし，もし，部分剰余R_iの演算の途中で，$R_i=[01.1]$bとなっても同じようなことが生じてしまいます．後で詳しく説明しますが，このような状態にならないように，式(2)を採用しながらも松原氏ら[(3)(4)]はディジット選択規則(**表1**)を提案しました．

では，式(1)を採用したらどうなるのでしょうか．$Y>X$のときに，Yの値を1ビット右にシフトする調節は不要です．はじめから1ビット左シフト($2R_i$)という余分なことはしていないからです．筆者は，偶然にも，除算・開平において数少ない式(1)を採用したGosling[(1)(2)]らの論文を参照し，検討していたので，後に式(2)の奇妙なことに気づくようになりました．式(2)を採用している本や論文の中では，$X>Y$の場合だけを想定して説明しているので，式(2)にあるこのような「ちょっとしたこと」が見過ごされています．

● キャリ・セーブ・アダーの演算結果をとらえる表現

表1のディジット選択規則の正しさや，もっと簡単な規則がないかを厳密に導いたり，証明するのは，実際にはなかなか難しい作業になります．そこで，部分剰余演算に用いるキャリ・セーブ・アダーの演算規則を記述したり，その動作を解析するために，**表2**に示す表現方式を考案しました．

2けたのキャリ・セーブ加算について考えます．キャリ・セーブ形式の各けたは，現在けたのサムsと下位けたからのキャリ入力cについて，$s+c=(s,\ c)$で数表現します．$2=(1,\ 1)$，$1=(0,\ 1)=(1,\ 0)$，$0=(0,\ 0)$です．加数部を行，被加数部を列で表し，演算結果を行と列が交差する部分で与えます．上位側けたの演算結果は，下位側けたからのキャリ入力により確定します．一方，下位側けたは，下位側けたへのキャリ入力があるときとないときがあるので，演算結果を記号＊＝{1, 2}または，記号＋＝{0, 1}で表現します．キャリ・セーブ・アダーの初期入力状態では，{0, 1}しかありませんので，**表2**(**a**)に記述しました．それから，**表2**(**b**)に記述したように，部分剰余演算では，ディジット選択値によって決まる加数部への各けた入力値は{0, 1}で，部分剰余の被加数部の各けた入力値だけ{0, 1, 2}のキャリ・セーブ形式で扱うことに気をつけてください．

それでは，これからキャリ・セーブ・アダーを用いる場合のディジット選択規則を導出していきます．形式ばった説明をしていくと分かりにくいので，まず，既に与えている**表1**のディジット選択規則について，演算長6ビットの場合の例1(**表3**)を用いて，正しく動作するのかを検証しながら説明していきます．

● 6ビット長の除算でディジット選択規則を検証してみる

表3の見方は，列に現在の部分剰余値R_iを，行に除数Xの$-q_i$倍である加数を与えて，行と列の交差する部分に左1ビット・シフトして，捨て去る最上位けたと続く上位3けた分のキャリ伝播が表現できるように結果を与えています．

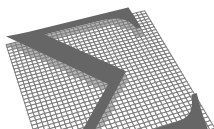



表3　例1: 演算長6ビットの除算
商ディジット選択と部分剰余演算結果．

q_i=1	110111	110110	110101	110100	110011	110010	110001	110000
000101	111020	111011	111010	111001	110120	110111	110110	110101
000102	111021	111020	111011	111010	110121	110120	110111	110102
000111	111110	111101	111020	111011	111010	111001	110120	110111
000112	111111	111110	111021	111020	111011	111010	110121	110120
001001	111120	111111	111110	111101	111020	111011	111010	111001
000201	111120	111111	111110	111101	111020	111011	111010	111001
001002	111121	111120	111111	111110	111021	111020	111011	111010
001011	000010	000001	111120	111111	111110	111101	111020	111011
000211	000010	000001	111120	111111	111110	111101	111020	111011
001012	000011	000010	111121	111120	111111	111110	111021	111020
001101	000020	000011	000010	000001	111120	111111	111110	111101
001102	000021	000020	000011	000010	111121	111120	111111	111110
001111	000110	000101	000020	000011	000010	000001	111120	111111
001112	000111	000110	000021	000020	000011	000010	111121	111120
010001							000020	000001
010011								000011

(**a**)q_i=1

q_i=0	000000
000000	000000
000010	000010

q_i= 0	000000
111210	000010
111200	000000
111110	111110
111100	111100
111010	111010
111000	111000

(**c**)q_i=0

q_i= -1	001000	001001	001010	001011	001100	001101	001110	001111
110210	000010	000011	000100	000101	000110	000111	001000	001001
110200	000000	000001	000010	000011	000100	000101	000110	000111
110110	111110	111111	000000	000001	000010	000011	000100	000101
110100	111100	111101	111110	111111	000000	000001	000010	000011
110010	111010	111011	111110	111101	111110	111111	000000	000001
110000	111000	111001	111010	111011	111100	111101	111110	111111
101200					111100	111101	111110	111111
101210					111110	111111	000000	000001
101110					111010	111011	111100	111101
101100					111000	111001	111010	111011
101020					111000	111001	111010	111011
101010					110110	110111	111000	111001

q_i∈{-1，0，+1}	加数(除数の-q_i倍)
被加数(部分剰余)	加算: 上位3ビットキャリ伝播結果

(**b**)q_i=－1

ディジットq_i∈{－1，0，＋1}の選択値によって分けています．**表3**においてq_i＝1を選択した場合，式(3)から，部分剰余R_iに－Xを加算することから，Xの反転＋(最下位けた1)を加算します．ここで**表2**の表現を使うことから，便宜上，R_i＋(最下位けた1)にXの反転を加算することにします．

例えば，R_i＝[000101]b，X＝[00.1000]bの場合，R_i＋(最下位けた1)＋(Xの反転)＝[000102]b＋[110111]b＝[111021]bとなります．また，R_i＝[001011]b，X＝[00.1001]bの場合，R_i＋(最下位けた1)＋(Xの反転)＝[001012]b＋[110110]b＝[111210]bとなりますが，結果を左1ビット・シフトして捨て去り，続く上位3けたをキャリ伝播させることから，$[r_{-1}r_0r_1r_2r_3r_4]$＝[111210]b＝[000010]bのようにキャリ伝播させて，最上位けたr_{-1}を捨て，R_{i+1}＝$[r_0r_1r_2r_3r_40]$＝[000100]bとします．ここで，左1ビット・シフトする場合，最上位けたr_{-1}と符号の連続性を保つために，r_{-1}＝r_0でなければならないことに気をつけてください．部分剰余の初期値R_0が[0010…]b，[0011…]bです．さらに，開平と逆開平では[0001…]bが加わります．初期演算状態部分を■で示しています．それら演算結果を左1ビット・シフトして，最上位けたを捨てて，部分剰余演算に新たに加わる入力値があれば追加して**表3**を完成させていきます．ただし，加数値の列ごとに到達する部分剰余値を追加していきます．そうすると，ディジット選択規則(**表1**)の範囲をはみ出すことなく，**表3**が完結しているのが分かります．つまり，演算長6ビットの場合にディジット選択規則(**表1**)が正しく動作していることが示されています．

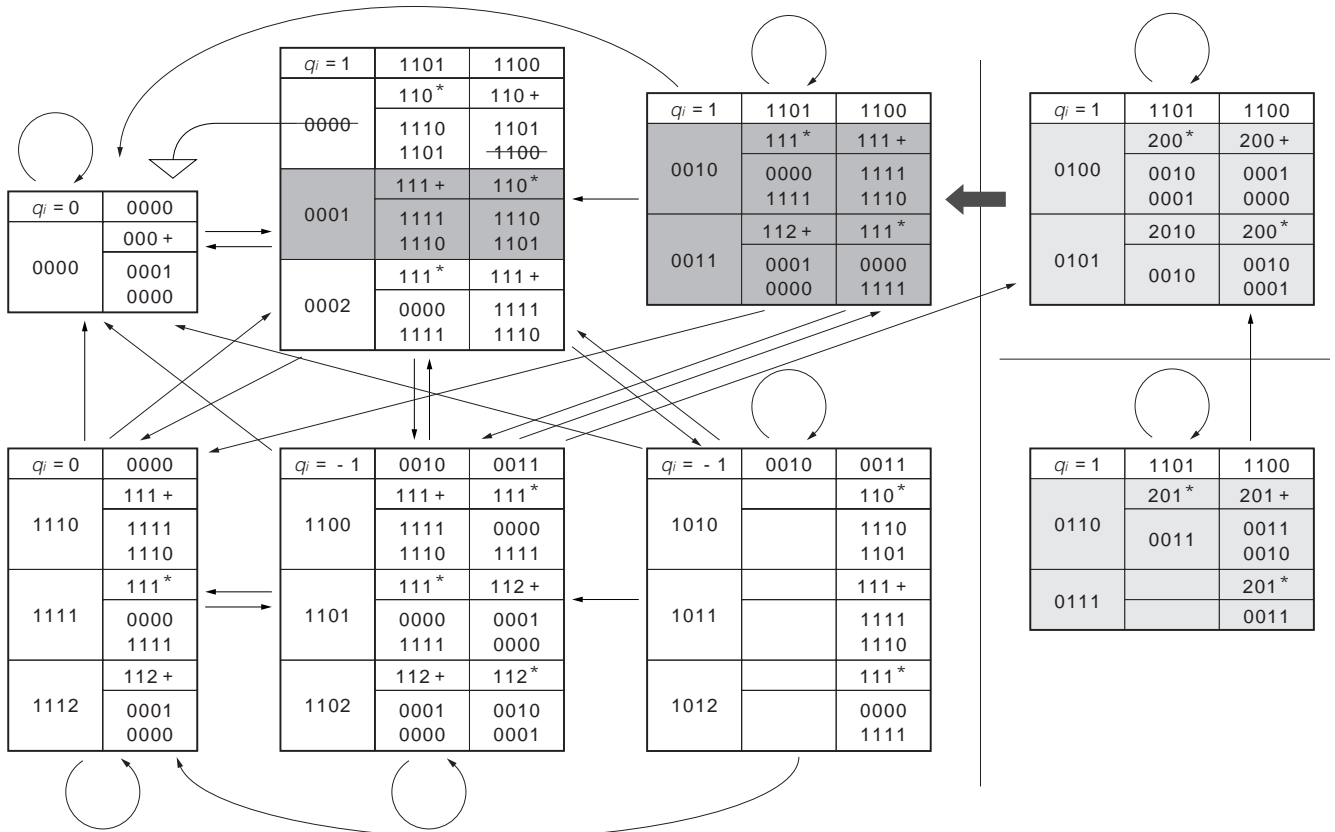



図1　ディジット選択規則の導出図
任意の演算長の場合にもディジット選択規則（表1）が正しく動作していることを示すために，上位4ビットだけで表現できるように表3をもとにして作成した図．

● 任意ビット長における演算動作を追跡してディジット選択規則を導出・検証

任意の演算長の場合にもディジット選択規則（**表1**）が正しく動作していることを示すために，上位4ビットだけで表現できるように作成した**図1**で考えます．**図1**は**表3**をもとに作成しています。演算結果の上位から4けた目は，下位けたからのキャリ入力の有り無しによって値が異なるので，**表2**で使った記号＋＝{0，1}または，＊＝{1，2}で示しています．また，部分剰余値が演算によって推移していく過程が分かるように，推移先を矢印で示しました．

ここで注意して欲しいのは，部分剰余初期値R_0＝{[0010…]b，[0011…]b，[0001…]b}から演算によって推移していく部分剰余値以外は表示してはならないということです．任意の部分剰余値からスタートすると，ディジット選択規則（**表1**）の範囲からはみ出してしまう場合があり，正しく動作することが証明できません．これが最適なディジット選択規則をエレガントに見つけることを困難にしていて，試行錯誤によって導出する必要があります．

例えば，**図1**には，もし部分剰余値が[0000]bのとき，q_i＝1を選択する規則が可能かを試した場合が示してあります．これはキャリ伝播した上位3ビットだけの参照で可能な解が存在するのかを試すために考えられるディジット選択規則の一つの候補です．

残念ながら，[0000]b＋[1100]bのときに，演算結果が[1100]bになる場合があり，左1ビット・シフトで次の部分剰余値が[100…]bになるため，ディジット選択規則からはみ出してしまい採用できません．それで，**図1**に示すように部分剰余値が[0000]bのときは，q_i＝0を選択するように**表1**のディジット選択規則を決めたのです．そして，初期部分剰余値からスタートしたすべての部分剰余値の演算結果の推移が**図1**の中で閉じていることから，任意の演算長の場合にも**表1**のディジット選択規則が正しく動作することが分かります．

ところで，**図1**には，$X>Y$を仮定して漸化式(2)を採用して，初期スタートする部分剰余値が[0101]b，[0110]b，[0111]bの場合も加えて示しており，これも正しく動作することを証明しています．

次に，キャリ伝播した上位3ビットだけの参照で可能な

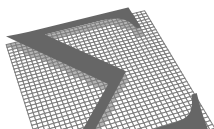
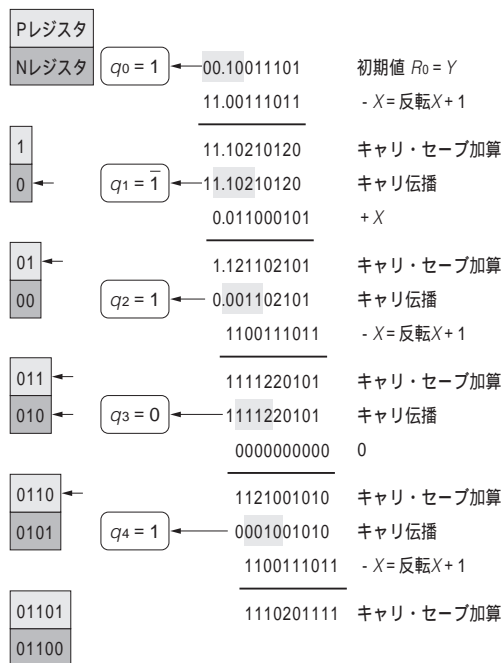



図2　除算の演算過程

$$Q=\sqrt{[0.1]\,\mathrm{b}}=[0.10110101]\,\mathrm{b}$$

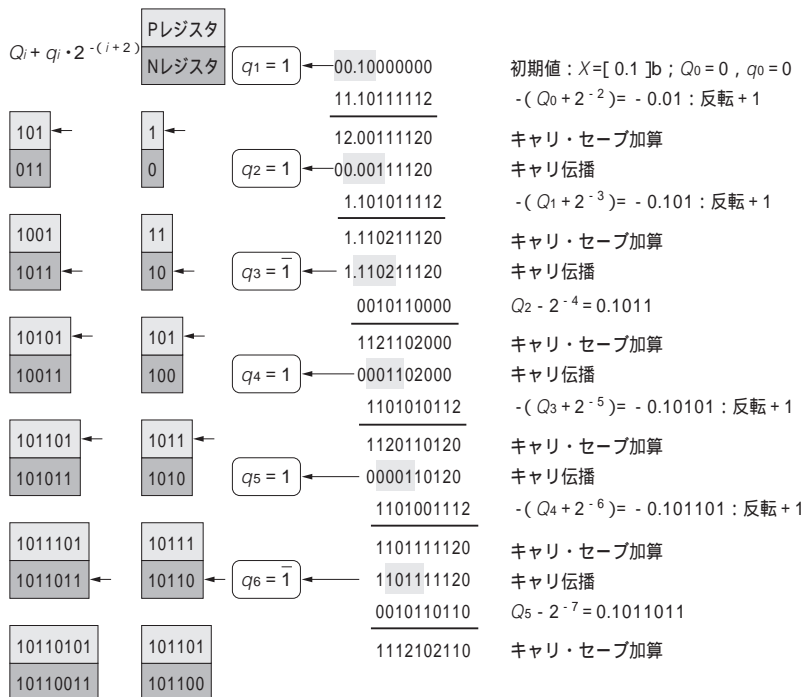



図3　開平の演算過程

解が存在するかということに関して，次の反例で，存在しないということを示します．部分剰余値が[000] bのときの候補として，$q_i=0$を選択する規則しか残っていませんが，演算長6ビットの例で，$Y=[00.1101]$ b，$X=[00.1000]$ bとします．

[001102] b＋[110111] b＝[112021] b＝[000021] b

[001] bから，$q=+1$，反転$X=[110111]$ b

[000210] b＋[000000] b＝[001010] b

[000] bから，$q=0$

[010101] b＋[110111] b＝[201020] b＝[001020] b

[010] bから，$q=+1$，反転$X=[110111]$ b

部分剰余値が[010201] bとなり，これ以降は選択規則の定義範囲をはみ出します．

これまで説明してきたことを具体的に理解してもらうために，除算，開平，逆開平の演算過程をそれぞれ例2(**図2**)と例3(**図3**)，例4(**図4**)で示します．

● 例2…除算の例

図2は除算の例です．on-the-fly 変換は，$q_i=\bar{1}$が選択されたときに，即座に2進数に変換できるように，Pレジスタと N レジスタの内容を準備しておき決定します．「←」はレジスタの新しい設定値として採用されたことを示しています．左1ビット・シフトのために最上位けたは捨て去りますが，次のけたと符号値が00または11のように連続していることに気をつけてください．

● 例3…開平の例

図3は開平の例です．除算のときに使うレジスタXには，部分開平値Q_iを設定します．詳しくは，$(Q_i+q_{i+1}\cdot 2^{-(i+2)})$のように余分な項$q_{i+1}\cdot 2^{-(i+2)}$が加わっているので，除算のときのon-the-fly変換の機能に追加して，同時に$(Q_i+q_{i+1}\cdot 2^{-(i+2)})$も2進数変換していかなければなりません．$q_{i+1}=1$ならば，$+2^{-(i+2)}$を加算します．下位に[01] bを追加することになります．$q_{i+1}=\bar{1}$ならば，$-2^{-(i+2)}$なので，上位の1を借りてきて，下位に[11] bを追加することになります．これはQ_iのon-the-fly変換を利用すれば簡単にできます．

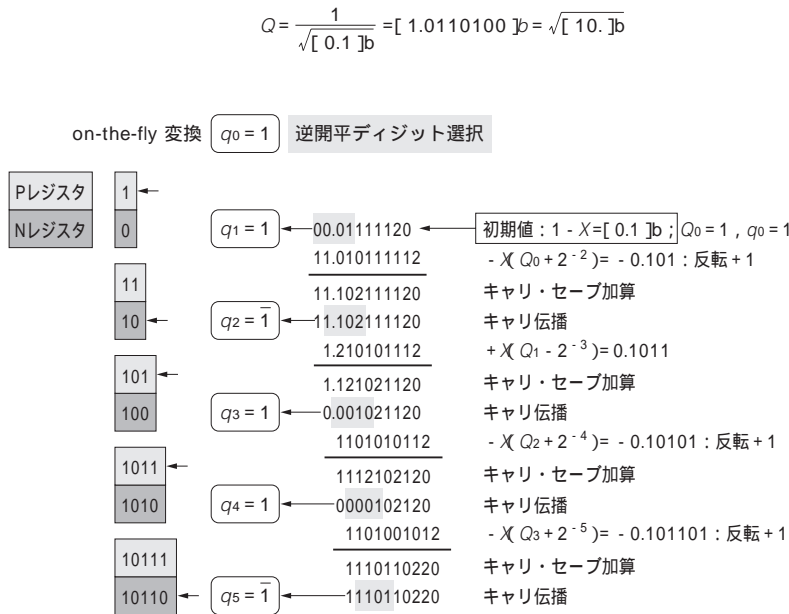



図4　逆開平数の演算過程

● 例4…逆開平数の例

図4は逆開平数の例です．YレジスタにXを設定します．$q_{i+1} \cdot X(Q_i + q_{i+1} \cdot 2^{-(i+1)})$の逐次乗算が必要なために，キャリ・セーブ・アダー2を使います．演算過程の詳細は，図の右側を参照してください．Zレジスタには，逐次乗算結果(＝XQ_i)が累積していきます．実際には，細かい設計が必要ですが，ここでは，原理を説明するのが目的なので，演算過程はある程度簡略化した設計で示しています．Xの逆開平数値は，Q_iに1ビットごとにon-the-flyで確定していきますが，同時に，ZレジスタにXの開平値が確定していくことにも気をつけてください．

符号付きディジット(signed-digit)アダー・ベースの設計

除算・開平・逆数・逆開平数の部分剰余演算のための加算器として，これまでキャリ・セーブ・アダーを用いた設計を基本に説明してきました．キャリ伝播のない加算器として，キャリ・セーブ・アダーの他に，**表4(b)**に示す符号付きディジット[signed-digit，冗長2進(redundant binary)とも呼ばれる]の演算規則に従う加算器があります．

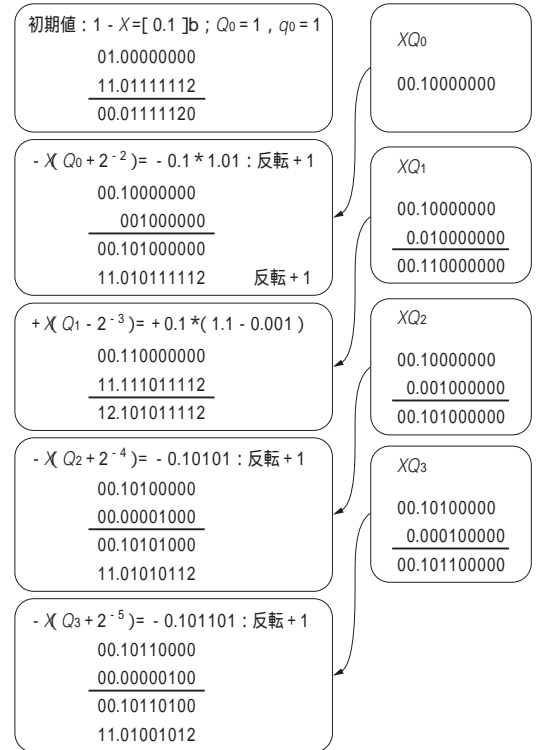



表4　キャリ伝播のない加算器

x+y	キャリ入力	キャリ出力	サム
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
0	1	0	1
1	1	0	2
2	1	1	1
3	1	1	2

(**a**) キャリ・セーブ・アダー：3-2カウンタの演算規則
x ∈{0，1，2}，y ∈{0，1}

x+y	キャリ入力	キャリ出力	サム
$\bar{1}$	0	0	$\bar{1}$
0	0	0	0
1	0	1	$\bar{1}$
2	0	1	0
$\bar{1}$	1	0	0
0	1	0	1
1	1	1	0
2	1	1	1

(**b**) 符号付きディジット：SSDアダーの演算規則
x ∈{$\bar{1}$，0，1}，y ∈{0，1}

● キャリ・セーブ・アダーと符号付きディジット・アダーを比較

比較のために，キャリ・セーブ・アダー(3-2カウンタ)と併記しています．また，**図5**に示すように，部分剰余演算に用いられる加算タイプは，一方の入力が{−1，0，＋1}で，もう一方が{0，1}なので，通常，符号付きディジット・アダーと呼ばれている両入力共に{−1，0，＋1}であるものとは異なることから，ここではSSDアダー(simplified signed-digit adder)と呼ぶことにします(**表5**)．SSDアダーは，現在のけた値が1のときには必ず上位けたにキャリを出し，サム値を−1にします(01→1$\bar{1}$)．下位けたから

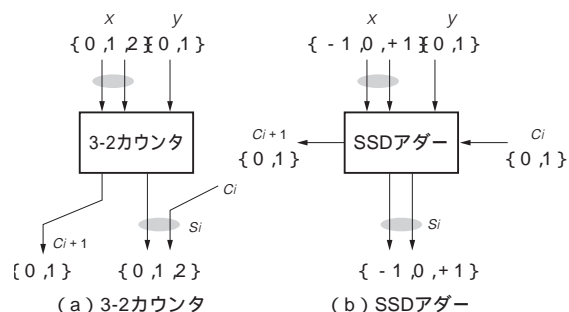
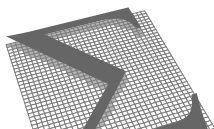



図5　二つの加算タイプのブロック図表現の比較
(a)と(b)について，数表現と入出力関係の違いを比較している．

図6　二つの加算タイプの論理回路図の比較
それぞれ回路数3個と5個の違いに注意．SSDアダーを2段で実現するために2個余分に必要になる．

表5　SSDアダーの演算結果(2桁分)

+ = {0, 1} − = {$\bar{1}$, 0}		加数 11	加数 10	加数 01	加数 00
被加数	1 1	1+	1−	0+	0−
	1 0	1−	0+	0−	$\bar{1}$+
	1 $\bar{1}$	0+	0−	$\bar{1}$+	$\bar{1}$−
	0 1	0+	0−	1+	1−
	0 0	0−	$\bar{1}$+	1−	0+
	0 $\bar{1}$	$\bar{1}$	$\bar{1}$−	0+	0−
	$\bar{1}$ 1	1+	1−	0+	0−
	$\bar{1}$ 0	1−	0+	0−	$\bar{1}$+
	$\bar{1}$ $\bar{1}$	0+	0−	$\bar{1}$+	$\bar{1}$−

表6　SSDアダーのディジット選択規則の導出
上位3けたのキャリ伝播で可能．

	q_i=1	加数 1.00	加数 1.01	q_i=$\bar{1}$	加数 0.11	加数 0.10	q_i=0	加数 0.00
被加数	010	0$\bar{1}$+	00−	0 $\bar{1}$ 0	01−	00+	000	000
	011	00−	00+	0 $\bar{1}$ $\bar{1}$	00+	00−	001	001
	100	00+		$\bar{1}$ 0 0	00−		00$\bar{1}$	00$\bar{1}$
	101	01−		$\bar{1}$ 0 $\bar{1}$	0$\bar{1}$+			

キャリ入力があってもサム値が0になるため，けた上げが生じません．そして，キャリ・セーブと同等の2段回路で実現するために，x入力側は(i_+，i_-，i_a)＝(1，$\bar{1}$，絶対値)の信号割り当てにします．そのため，**図6**に示すように，SSDアダーはキャリ・セーブ・アダー回路に比べて回路規模が大きくなるのですが，部分剰余演算では，以下に説明するようにディジット選択規則が対称になり，導出が容易になることを示します．

● ディジット選択規則が対称になるSSDアダーの方がディジット選択規則の導出が簡単

表6に示すように，部分剰余演算にSSDアダーを用いる場合，ディジット選択規則は上位3けたのキャリ伝播だけで可能なことが，演算結果の最上位けたがすべて0であることから，左1ビット・シフトしてもまた同じ表で表現でき閉じていることから簡単に導出できます．

キャリ・セーブの場合は，2の補数表現の符号けたに2^1けたを当てていましたが，SSDの場合は，数の表現自身に符号をもっていることから，符号けた(2^1けた)は不要です．ただし，加数Xは{0，1}だけを要素とする数列であることから，減算する場合は，$-X$の2の補数表現を用いて行います．つまり，反転X＋(最下位けた＋1)を加算します．例えば，$X＝0.11\cdots$のとき，$-X＝[1.00\cdots]$b＋[0.00$\cdots$1]bです．

次回は，除算・開平・逆数・逆開平数演算を加速する方法について，いくつかのアイデアを取り上げ紹介していきます．

参考・引用*文献

(1) J. B. Gosling and C. M. S. Blakeley；Arithmetic unit with integral division and square root，IEE Proceedings Vol.34，Pt.E，No.1，pp.17-23，Jan. 1987.
(2) J.H.P.Zurawski and J.B.Gosling；Design of a High-Speed Squre Root Multiply and Divide Unit，IEEE Trans. On Computers，Vol.C-36 No.1，pp.13-23，Jan.1987.
(3) 松原玄宗，井出進博，鈴木清吾；非同期回路を用いたハード共用型SRT除算/平方根演算器，信学技法，DSP95-1-1，ICD95-150，pp.29-36，電子通信学会，1995年10月.
(4) G. Matsubara，N. Ide，H. Tago，S. Suzuki and N. Goto；30-ns 55-b Shared Radix 2 Division and Square Root Using a Self-Timed Circuit，Proc. Of 12th IEEE Computer Arithmetic Symp，pp.98-105，Jul.1995.

とのむら・もとのぶ
大日本印刷株式会社　電子デバイス事業部　電子デバイス研究所